{%@define j=Jamal%}{%j%} Microsoft Word Integration Module

This module supports the use of {%j%} macros in Microsoft Word documents (docx format files only). The macros can be embedded into the text of the Word document and the output will be another word document with the macros processed. During the processing the text formatting will be preserved as much as it is possible.

The module uses special techniques to invoke the {%j%} processor. {%j%} is a text processor.

A Word document, on the other hand mixes textual and formatting information.

Before discussing the details of the technique, let's look at the structure of a Word document.

* A Word document is a collection of paragraphs and tables intermixed.
* A paragraph is a collection of "runs".
* A paragraph contains formatting information, which describes the paragraph's appearance. This is like indenting, centering, etc.
* A run contains text segments and formatting information.
* The formatting does not change inside a run, and usually there is only one text segment in a run. The formatting information of the run describes the appearance of the text.
* Tables intermix with paragraphs. Tables contain cells and the cells contain paragraphs. Paragraphs in table cells may intermix with tables any level deep.

The module uses the following techniques to process the macros:

1. It reads the document using Apache POI into the memory and then starts to process the paragraphs and the tables.
2. The module implements the Input API interface in a very special way. This implementation stored the characters of a single run in a buffer and if the processing needs more characters, it dynamically fetches them from the subsequent runs.
3. When the processing does not need more characters the {%j%} processor does not ask for more characters, and it stops. At this stage the module restarts the processor in case there is more text to process. This way the processing of the document is done in chunks. When a chunk is processed the result of the processing replaces the paragraphs and the runs used up as input. The formatting of the very first run of the chunk is preserved and used for the whole text of the formatting.

Because the way the processing is done there are a few special rules that macros should follow in a Word document.

* A chunk cannot extend from a top-level paragraph into a table and cannot extend out of a table cell. A macro starting in a top level (not in table) paragraph should be closed before the next table if any.
* A macro starting in a paragraph inside a table cell should be closed inside the cell.
* If there is any table nested in a table cell, then a macro starting in a cell paragraph should be closed before the nested table.
* Any formatting inside the definition of a user defined macro will be lost and not copied to the output at the place where the macro is used. The output of the `define` macro is an empty string. A user defined macro in {%j%} is a processable text including replacements for the actual values of the parameters. It does not contain any formatting.
* Any formatting in the actual value of the parameters of a macro will be lost and not copied to the output. The reason is the same as for the user defined macro body. The parameter of a user defined macro or the input of a built-in macro is text without formatting.

The macro `defer` will work very different in the case of a docx file. This schedules the execution of some macros after the processing is done with the whole input. These macros also get the whole processed input as text in the macro, named `$output`. Processing a docx file happens in chunks and the processor is closed after each chunk. It means that any deferred macros will be executed after the chunk defining it is closed. The processed output will contain only the one chunk that was processed.

The restrictions of the processing {%j%} in a docx file are not source from the implementation of the module. These essentially come from the fact that a docx file contains the text and the formatting mixed. It is not possible to process the whole text as a single chunk and keep the formatting. In case of complex macros there is no clear mapping between the input and the output text. Without a mapping between the characters of the input and the output there is no way to copy the formatting.

*Using the module*

The module can be used in two ways.

* Use the command line version of the {%j%} using the –docx option.
* Call the exported XWPFProcessor API.

*Dependency*

To use the module, you have to have the code on the classpath. This can be done adding

{%@snip:xml pom=../../../pom.xml%}{%#define VERSION={%pom /project/version/text()%}%}

**<groupId>com.javax0.{%j%}</groupId>**

**<artifactId>{%j%}-word</artifactId>**

**<version>{%VERSION%}</version>**

to your Maven pom.xml.